Laboratory 3, Motor control

Last update: 25-03-2024 Hash: 47b0b55

1 Introduction

Purpose of this experience is to implement a PID controller to control the motors of the TBot.

2 Preliminary Information

2.1 Data logger

The TBot is equipped with a data logger that can record almost any type of data. There are two available communication interfaces:

- Serial Datalogger (via USB debug interface)
- WiFi Datalogger

2.1.1 MATLAB

The serial datalogger can be invoked from MATLAB using the command:

```
data = serial_datalog(COM_port , packet_specification)
```

where `COM_port` is the port of the serial datalogger (e.g., COM3) and `packet_specification` is a cell array of char arrays that specifies the type of data to be recorded. The port currently in use can be immediately detected by invoking the `seriallist` routine from the MATLAB Command, provided that the STM32F7 board is the only device connected to the host computer. If so, the routine should return a list of two items, of which the first is the COM1 port reserved by the system for internal uses, and the other is the port used by the STM32F7 board to communicate with the host computer.
The data received from the robot are simultaneously displayed on a MATLAB figure in real-time and saved into a memory buffer. Once the figure is closed, the buffer content is copied into the output variable `data`, which is a MATLAB structure with the following fields:

- `data.time` is the field containing the time instants at which the data samples have been transmitted, with respect to the beginning of the data logging process.
- `data.out` is a cell array containing the data samples sent by the balancing robot. Each element of the cell array corresponds to a different logged signal. It is a matrix, whose columns are the values of the logged signal. Therefore, the number of rows is equal to the size of the signal (number of components), and the number of columns is instead equal to the number of received data samples. The number of received data samples never exceeds the size of the buffer used to hold the data during the data logging process (by default, the buffer size is equal to 1000). For example, the command `y = data.out{2}(2,:)` retrieves the 2nd component of the 2nd signal sent by the STM32F7 board.

For example, by invoking the command:

```
data = serial_datalog('COM8',{'2*single','2*single'}, 'baudrate', 115200)
```

the data logger will expect to receive a total of four single precision floating point numbers, which will be displayed in two separate sub-plots.
To use the WiFi datalogger, the following command should be invoked:

```
data = udp_datalog(ip, port, packet_specification)
```

where `ip` is the IP address of the robot, `port` is the port number used by the WiFi datalogger, and `packet_specification` is a cell array of char arrays that specifies the type of data to be recorded.
For example, you can invoke the command:

```
data = udp_datalog('147.162.118.61',9090, {'2*single','2*single'})
```

ATTENTION: every turtlebot has a different IP address. Please check (or ask for it) the IP address of the robot you are trying to connect to.

If something hangs or crashes during the data logging process and nothing is working anymore, you can try those two commands:

```
clear all
instrreset
```

2.1.2 STM32

On the STM32 side, the data logger must be properly configured. Example code is provided in the project for Lab 2.
You can choose which communication interface you want to use by calling the following instructions inside the main function (these function calls are already included in the project):

```
//logger.uart_handle = huart3; // for serial
logger.uart_handle = huart2; // for wifi
```

The basic steps to initialize and run the data logger are explained in the code below:

```
/* declare an ertc_dlog struct */
struct ertc_dlog logger;
.
.
.
int main(void)
{
    .
    .
    .
    /* USER CODE BEGIN 2 */
    /* select the correct UART: uart3 is the default peripheral used */
    logger.uart_handle = huart3;
    // logger.uart_handle = huart2; // for wifi

    /* in this example, we use the timer to periodically send data through the data logger */
    HAL_TIM_Base_Start_IT(&htim6);
    /* USER CODE END 2 */
    .
    .
    .
    while (1)
    {
        /* USER CODE BEGIN 3 */
        /* continuously update the state of the data logger */
        ertc_dlog_update(&logger);
    }
    /* USER CODE END 3 */
}
```

To send arbitrary data, it is necessary to define a struct as follows:

```
struct datalog
{
    float w1, w2;
    float u1, u2;
} data;
```

The structure above is just an example; the structure can be arbitrary, but it is important to put the `datalog` attribute after the word `struct`.
To send the data:

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM6)
    {
        data.w1 = 10;
        data.w2 += 1.085;
        data.u1 = -3.14;
        data.u2 = 0.555683;
        ertc_dlog_send(&logger, &data, sizeof(data));
    }
}
```

To avoid the case in which the board starts sending "useful" data before MATLAB is ready to show them, it is suggested to check for the value of `tx_enabled` before setting the value of the reference signal. `tx_enabled` indicates whenever the data-logger, from the PC side, is able to receive data.

WARNING: To avoid excessive transmission time and delays please send only numeric data (e.g., float, int, uint8_t, etc.). Do **not** send strings.

3 Exercises

3.1 Exercise 1

- The main goal for this laboratory is to implement a PI controller to control the angular speed of a DC motor.
- The reference signal should be an angular speed, measured in rounds per minute [rpm] or radians per second [RAD/s].
- The controller should be able to manage negative reference values.
- The reference values, one for each motor, can be hardcoded; Bonus: implement a functionality that allows setting the reference values using the keypad.
- Default values for the k_p and k_i constants have to be found by trial and error.
- **Implement the control using the *Forward/Brake* technique.**
- All the steps needed to accomplish the goal, described in the document regarding motor control, should be executed from inside the `HAL_TIM_PeriodElapsedCallback` function, which, as previously mentioned, is a callback function called when a timer's period is elapsed. Remember to check which timer caused the callback to be called: in particular, we are interested in handling the events raised by the timer TIM6.
- Use the data logger to save the relevant information, like the step response of the closed-loop system.
- **All the relevant information about the HAL functions, code structure, and computation steps are provided in the document of the lesson regarding motor control.**

3.2 Exercise 2

- Implement the controller with the *Forward/Coast* technique and compare the results with those obtained in Exercise 1.

3.3 Bonus

- Implement the Anti-windup feature of the PI controller.